ICASE

ON THE MAPPING PROBLEM FOR THE

FINITE ELEMENT MACHINE

Shahid H. Bokhari

Report Number 79-16

July 24, 1979

INSTITUTE FOR COMPUTER APPLICATIONS IN SCIENCE AND ENGINEERING

NASA Langley Research Center, Hampton, Virginia

Operated by the

UNIVERSITIES SPACE USRA RESEARCH ASSOCIATION

ON THE MAPPING PROBLEM FOR THE FINITE ELEMENT MACHINE

Shahid H. Bokhari
Institute for Computer Applications in Science & Engineering (ICASE)
MS 132C, NASA Langley Research Center, Hampton, VA 23665

Abstract--In array processors it is important to map problem modules onto processors such that modules that communicate with each other lie, as far as possible, on adjacent processors. This mapping problem is formulated in graph theoretic terms and shown to be equivalent, in its most general form, to the graph isomorphism problem. The problem is also very similar to the bandwidth reduction problem for sparse matrices and to the quadratic assignment problem.

It appears unlikely that an efficient exact algorithm for the general mapping problem will ever be found. Research in this area must concentrate on efficient heuristics that find good solutions in most cases. A heuristic algorithm that proceeds by sequences of pairwise interchanges alternating with probabilistic jumps is described. This algorithm has been used to solve practical mapping problems on a specific array processor (the Finite Element Machine) with good results. Results for a set of practical problems are tabulated, several of which are illustrated.

I. Introduction

Most arrays of processors are incompletely connected, that is, a direct link does not connect each pair of processors. The reasons for this include (1) the fact that the total number of links in completely connected systems increases as the square of the number of processors--a growth rate that is unacceptable in most cases, and (2) the number of input/output ports on each individual processor increases linearly with the number of processors--this is usually not possible because the number of I/O ports is generally fixed at some constant value.

Suppose a problem made up of several modules that execute in parallel is to be solved on an incompletely connected array. When assigning modules to processors, pairs of modules that communicate with each other should be placed, as far as possible, on processors that are directly connected. We call the assignment of modules to processors a mapping and the problem of maximizing the number of pairs of communicating modules that fall on pairs of directly connected processors the Mapping Problem.

This research was supported by NASA Contracts NAS1-14101 and NAS1-14472 while the author was resident at ICASE.

In this paper we first show that the problem of finding the best mapping is, in general, very difficult. We then describe a heuristic algorithm that has been developed to solve this problem for a specific array processor. We start by giving a mathematical formulation of the problem in Section II. In Section III we show that in its most general form, the mapping problem is equivalent to the graph isomorphism problem, one of the classical unsolved combinatorial problems. We point out the similarities between the mapping problem and the bandwidth reduction and quadratic assignment problems. Exact solutions for neither of these problems exist and they are solved approximately using heuristic algorithms.

In Section IV we describe how the mapping problem arises when solving structural problems on the Finite Element Machine (FEM), an array of processors currently under development at NASA Langley Research Center. In Section V we describe a simple heuristic algorithm that has been implemented and used to find mappings for the finite element machine with very encouraging results. Results for a number of test cases are tabulated, several of which are illustrated.

II. Mathematical Formulation

Let the graph of the problem to be mapped onto the array be denoted $G_p=<V_p,E_p>$, where the nodes or vertices V_p correspond to the set of modules and each edge $(x,y)\in E_p$ denotes that modules $x,y\in V_p$ communicate with each other.

Let the graph of the array processor be denoted $G_a=<V_a,E_a>$, where V_a is the set of processors and the edges E_a represent the interconnection pattern of the processors.

The problem graph G_p may be considered to be a set of vertices V_p and a function $G_p:V_p \times V_p \rightarrow \{0,1\}$, such that $G_p(x,y)=G_p(y,x)$ and $G_p(x,x)=0$ for all $x,y\in V_p$. $G_p(x,y)=1$ is taken to mean that there is an edge between x and y, i.e. the pair $(x,y)\in E_p$.

The graph of the array, G_a, may similarly be considered a set of vertices V_a and a function $G_a: V_a \times V_a \rightarrow \{0,1\}$.

We assume that $|V_p| = |V_a|$. If $|V_p| < |V_a|$, a suitable number of dummy vertices may be inserted into the problem. We do not consider the case $|V_p| > |V_a|$.

A mapping of problem modules onto processors is denoted by the function $f_m: V_p \xrightarrow[\text{onto}]{1\text{-}1} V_a$.

The quality of a mapping is determined by the number of problem edges that fall on array edges. We call this number the cardinality of the mapping, denoted $|f_m|$.

The cardinality of a mapping f_m is

$$|f_m| = \frac{1}{2} \sum_{\substack{x \in V_p \\ y \in V_p}} G_p(x,y) * G_a(f_m(x), f_m(y)).$$

This formula arises as follows. $G_p(x,y) = 1$ if x and y in the problem graph are connected by an edge. $f_m(x)$ [$f_m(y)$] represents the processor onto which problem module x [y] is mapped. The expression $G_a(f_m(x), f_m(y)) = 1$ only if the processors onto which x and y are mapped are connected. Thus the expression inside the summation sign is 1 only if an edge connecting two modules falls on an edge connecting two processors. In summing over all $x \in V_p$ and $y \in V_p$ each processor edge is counted twice, hence the multiplying factor.

To find the best mapping, we must choose a function f_m that has maximum cardinality from among the $(|V_p|)!$ possible functions.

III. Problem Equivalences

In this section we show that the mapping problem, in its most general form (i.e. given arbitrary G_a and G_p), is computationally equivalent to the graph isomorphism problem. We point out the strong similarities between the mapping problem and the bandwidth reduction and quadratic assignment problems.

Graph Isomorphism

Two graphs G_1 and G_2 are said to be isomorphic to each other if there is a one-to-one correspondence between their vertices and between their edges such that the incidence relationships are preserved [1]. This may be stated more formally as follows: two graphs $G_1: V_1 \times V_2 \rightarrow \{0,1\}$ and $G_2: V_2 \times V_2 \rightarrow \{0,1\}$ with $|V_1| = |V_2|$ are isomorphic if there exists a function $e: V_1 \xrightarrow[\text{onto}]{1\text{-}1} V_2$ such that $G_1(x,y) = G_2(e(x), e(y))$ for all $x, y \in V_1$ [2].

The problem of determining whether two graphs are isomorphic is one of the classical unsolved combinatorial problems. Exact efficient (i.e. polynomial time) algorithms for solving this problem for arbitrary graphs are not known although numerous researchers have attacked this problem [3]. Some researchers have reported success with heuristic algorithms applied to various restricted classes of graphs (see, for example [4]). It appears unlikely that an polynomial time solution to the general problem will ever be found.

We now show that if we had an exact algorithm for solving the general mapping problem, we would also be able to solve the graph isomorphism problem.

If two graphs are isomorphic, they must have the same number of edges and thus a function mapping G_1 onto G_2 and having cardinality equal to the total number of edges must exist. If we had an exact algorithm for solving the mapping problem, we could use it to map G_1 onto G_2 and, if the two were isomorphic, obtain a mapping of cardinality equal to the number of edges. Thus we could answer "yes" or "no" to the question "Are G_1 and G_2 isomorphic?" in polynomial time, for arbitrary G_1 and G_2, if we could solve the mapping problem in polynomial time for arbitrary G_1 and G_2. The mapping problem is therefore computationally equivalent to the graph isomorphism problem and we do not hold much hope for finding an exact polynomial time algorithm for its solution.

Bandwidth Reduction

The bandwidth reduction problem requires the permutation of the rows and columns of a sparse square matrix so as to cluster the non-zero entries as closely as possible about the main diagonal [5]. The mapping problem, as will become clear in the following sections, entails permuting the rows and columns of the adjacency matrix of a problem graph so that it resembles as closely as possible the adjacency matrix of the graph of the array of processors. Arrays of processors that have a regular interconnection pattern (as does the FEM), usually have an adjacency matrix composed mostly of several well-defined bands. The mapping problem for such arrays entails permuting the input matrix so that as many entries as possible fall on the bands. The similarity with bandwidth reduction is obvious.

The bandwidth reduction problem is known to be NP-complete [6]. Many heuristic algorithms for this problem have been developed [5],[7].

The Quadratic Assignment problem

In this problem we are given (1) a set of n objects alongwith a cost matrix in which each entry c_{ij} is a measure of the affinity between objects i and j, and (2) a set of n locations with a distance matrix in which entry d_{st} stands for the distance between locations s and t. A function p that maps objects onto locations is called an assignment. The problem of finding the assignment that minimizes $\sum_{i,j} c_{ij}d_{p(i)p(j)}$ is called the quadratic assignment problem[8]. This problem is exemplified by the task of locating electrical assemblies in given slots so as to minimize the total length of interconnecting wires. No efficient algorithm for the solution of this problem is known.

If the affinity and distance matrices be symmetric and have 0,1 entries, the quadratic assignment problem reduces to the mapping problem.

IV. The Finite Element Machine

The Finite Element Machine (FEM), presently under development at NASA Langley Research Center, is an array of microcomputers interconnected in an "eight-nearest neighbor" interconnection pattern (Fig. 1) [9],[10]. In addition to the nearest neighbor links, which are dedicated to communication between specific pairs of processors, there is a time shared global bus (not shown in Fig. 1) which is used for communication between pairs of nodes that are not adjacent.

The machine is to be used to solve structural analysis problems as follows. The structure is first reduced to a combinatorial graph. The edges of the graph correspond to structural members and the nodes to meeting points of the members (Fig. 2). Each node is assigned to a processor of the FEM and computation proceeds in parallel, as described by Jordan [9]. During the course of the computation, there is communication between pairs of processors only if the structural nodes mapped on them are connected in the physical problem. Thus, should an edge of the physical problem fall on an edge of the FEM, communication proceeds with greatest efficiency via the dedicated nearest neighbor connection. Should this not be the case, interprocessor communication must employ the time-shared global bus with consequent degradation in performance.

Mapping Structures onto the FEM

Fig. 3 shows the adjacency matrix of a 6 X 6 FEM. One possible way of mapping the problem structure of Fig. 2 onto the FEM would be to map node i of the structure onto node i of the FEM. This mapping is indicated in the adjacency matrix of the structure (Fig. 4) by the use of '*'s where an edge of the structure falls on an edge of the FEM, and '0's otherwise. The cardinality of this mapping is 32, while the total number of edges is 80.

We can attempt to increase the cardinality of the mapping by renumbering the nodes of the problem or, equivalently, permuting rows and columns of the adjacency matrix of the problem. The bottom part of Fig. 4 shows an improved mapping, with cardinality 74, obtained by applying the mapping algorithm that will be described in the following section . The permuted row and column labels indicate the renumbering that must be done to the nodes of the problem in order to obtain this improved mapping.

V. MAPPER: A Pairwise Interchange Algorithm

We have developed a heuristic algorithm that accepts as input the adjacency matrix of a problem graph and outputs a permutation of this matrix that matches more closely the adjacency matrix of the FEM.

The algorithm proceeds by sequences of pairwise interchanges, alternating with probabilistic jumps. It starts by accepting the problem matrix and the size of the square FEM onto which it is to be mapped. It then generates the adjacency matrix of the FEM and uses this for comparison while improving the mapping.

Most of the following listing is self explanatory. The function CARDINALITY(MAT) returns the cardinality of the mapping defined by the matrix MAT.

```
program MAPPER;
var MAT, BEST: adjacency_matrix;
    DONE, FLAG: boolean;
begin
   input adjacency matrix of problem, MAT;
   {MAT is taken to be the initial mapping}
   input the size of the FEM, n;
   {the FEM is an n X n array}
   generate adjacency matrix for n X n FEM;
   BEST:=MAT; {the best mapping found so far}
   DONE:=false;
      while not DONE do
      begin{MAIN}

         repeat{SEARCH}
            FLAG:=false;
            for each node do

               begin{AUGMENT}
                  1: examine the pairwise
                     exchange of this node
                     with all other nodes;
                  2: select the one which leads
                     to the largest gain in the
                     cardinality of the mapping;
                  3: if largest gain>=0 then
                     make the exchange;
                  4: if largest gain>0 then
                     FLAG:=true;
               end;{AUGMENT}

         until FLAG=false; {end SEARCH}

         if CARDINALITY(MAT)<CARDINALITY(BEST)
            then DONE:=TRUE
         else
           begin{JUMP}
              BEST:=MAT;
              randomly interchange
              n pairs of nodes of MAT;
           end;{JUMP}

       end;{MAIN}
    output BEST;
end.
```

The block SEARCH of this algorithm attempts to improve the mapping by considering all possible pairwise exchanges of node numberings. The exchange that leads to the maximum increase in cardinality of the mapping is made and the process AUGMENT repeated until no further gains are possible. At this point we leave SEARCH and if the mapping found during this pass through SEARCH is better than the best mapping found so far, a probabilistic jump is applied to the mapping and the algorithm returns to block SEARCH.

Pairwise interchanges are not guaranteed to lead to the best mapping and sometimes lead to mappings that are "dead ends" in that they are not very close to optimal and no pairwise exchange can improve them. The algorithm attempts to leave such dead ends by probabilistically "jumping" to nearby mappings that may permit improvement via pairwise interchanges.

The following is a detailed discussion of various aspects of the algorithm.

1. When carrying out pairwise exchanges, we choose for each node the exchange that leads to the largest gain in cardinality rather than the first gainful exchange encountered. We have found that this strategy leads to mappings that are consistently better than those obtained using the second criterion.

2. We make an exchange even if the largest gain encountered is zero. This has little effect at the outset, when interchanges with nonzero gains are easily found. Towards the end of the algorithm, this criterion helps slide past "dead ends" to mappings which, although they have the same cardinality, may permit further improvement.

3. If the number of nodes on the FEM is N=n X n, then the execution of block AUGMENT will take $O(N^2)$ time.

4. The algorithm will exit block SEARCH if no pairwise exchange leads to an improvement. If the cardinality of the mapping found during this pass through SEARCH is better than the one found during the last pass, the algorithm executes JUMP. Here it tries to break out of the "dead end" from which no pairwise exchange leads to an improvement by probabilistically jumping to a nearby mapping, which, although it will almost certainly have poorer cardinality, may lead to a better mapping upon further application of block SEARCH. A copy of the old mapping is saved in BEST, in case the new mapping is poorer.

5. The probabilistic jump described above needs to be far enough from the current mapping to offer the prospect of improvement, but not so far as to undo all the gains made up to this point. We have found that an interchange of n randomly selected pairs of nodes gives the best results.

6. If the mapping found after attempting to augment from a probabilistically disturbed mapping is poorer, the algorithm terminates. We have found that further probabilistic jumps very rarely lead to improvements.

7. For an N node FEM, the cardinality of a mapping cannot exceed 4N. Each pass through loop MAIN must lead to a gain of at least 1. The time required to execute this loop is dominated by AUGMENT, which takes $O(N^2)$ time. The algorithm thus takes $O(N^3)$ time in all.

VI. Performance of the Algorithm

The algorithm has been implemented and tested on about 20 structural problems of 9 to 49 nodes for FEMS of sizes 4 X 4 to 7 X 7. The results are tabulated in Table I. Some of these cases are illustrated in Figs. 5-7. In most cases the algorithm is able to improve the mapping dramatically.

It is difficult to say just how close to optimal the mappings obtained by the algorithm are, since we have no way of knowing what the best mapping for a specific probem is. In cases where the cardinality of the final mapping is close to the total number of edges, we can be sure that it is very close to optimal. For example, the mapping of Fig. 2 was improved from 32 to 74 as shown in Fig. 4. The final mapping is very close to the total number of edges (80) and must therefore be very near optimal. (For this specific example, it is possible to prove that the optimal mapping is of cardinality 78 [11]). In general, we have found that graphs whose input mappings have cardinalities of around 50 percent of the total number of edges or less can usually be improved dramatically.

To get a better idea of the performance of the algorithm, we mapped random permutations of the FEM onto itself. Since the FEM can be mapped perfectly onto itself, the success of the algorithm in doing so gives us some idea of how well it performs on general problems, for which it is impossible to specify the cardinality of the best mapping. The results of this experiment, in which we fed the algorithm 100 random permutations of 5 X 5 and 6 X 6 FEMS are listed in Fig. 8. Histograms for the initial cardinality, cardinality at the end of the first application of SEARCH and the cardinality at the termination of the algorithm are given (the difference between the latter two illustrates the impact of jumping). The algorithm is seen to perform very well in these experiments, suggesting that the results obtained when the algorithm is run on natural structural problems are also of similar quality.

The run times of the implemented algorithm on a CDC Cyber 175 vary from about 1/3 sec. for 4 X 4 problems to 30 sec. for 7 X 7 problems.

VII. Conclusions

The observed run times of our algorithm are quite acceptable for the current prototype 6 X 6 FEM. However, as the growth rate of time is bounded from below by N^2, the algorithm will probably not be suitable for very large arrays (say 32 X 32). For such arrays, entirely different heuristics will need to be developed.

VIII. Acknowledgements

The author is indebted to Dr. Robert Voigt for his constant encouragement of this research. Discussions with Harry Jordan, Wynne Calvert, David Loendorf and Patricia Sawyer have contributed significantly to this research. The author is grateful to Winifred Stalnaker for providing test problems from the National Transonic Facility, and to Brian Thoma for assistance in preparing the manuscript.

IX. References

[1] N. Deo, Graph Theory with Applications to Engineering and Computer Science, Englewood Cliffs: Prentice-Hall, (1974).

[2] R. M. Karp, "Probabilistic Analysis of a Canonical Numbering Algorithm for Graphs," Computer Science Division, Dept. EECS, U. of California, Berkeley, 1978.

[3] D. Corneil and R. C. Read, "The Graph Isomorphism Disease," Journal of Graph Theory, vol. 1, no. 4, Winter 1977, pp. 339-363.

[4] V. M. Kureichik and T. A. Bickart, "An Isomorphism Test for Homogeneous Graphs," Proc. 1979 Johns Hopkins Conference on Information Sciences and Systems, pp. 60-64.

[5] E. Cuthill, "Several Strategies for reducing the Bandwidth of Matrices," in Sparse Matrices and their Applications, Rose & Willoughby, eds., New York: Plenum press, 1972, pp. 157-166.

[6] R. E. Tarjan, "Graph Theory and Gaussian Elimination," in Sparse Matrix Computations, Bunch & Rose eds., New York: Plenum Press, 1976, pp. 3-22.

[7] N. E. Gibbs, W. G. Poole and P. K. Stockmeyer, "An Algorithm for Reducing the Bandwidth and Profile of a Sparse Matrix," SIAM J. Numerical Analysis, vol. 13, no. 2, April 1976, pp. 236-250.

[8] M. Hannan and J. M. Kurtzberg, "A Review of the Placement and Quadratic Assignment Problems," SIAM Review, vol. 14, no. 2, April 1972, pp. 324-342.

[9] H. Jordan, "A Special Purpose Architecture for Finite Element Analysis," Proc. 1978 Conf. on Parallel Processing, August 1978, pp 263-266.

[10] H. Jordan, M. Scalabrin and W. Calvert, "A Comparison of Three Types of Multiprocessor Algorithms," Proc. 1979 Conf. on Parallel Processing, August 1979.

[11] H. Jordan, personal communication.

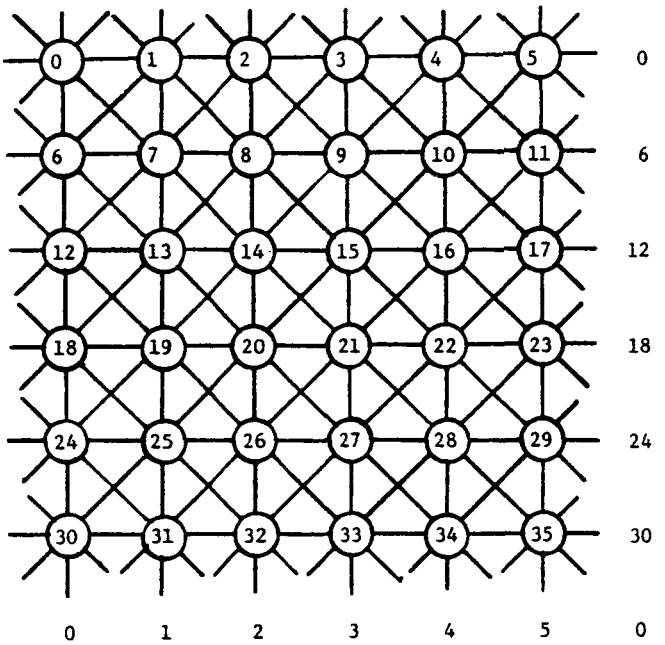


Fig. 1 A 6 X 6 Finite Element Machine (FEM)

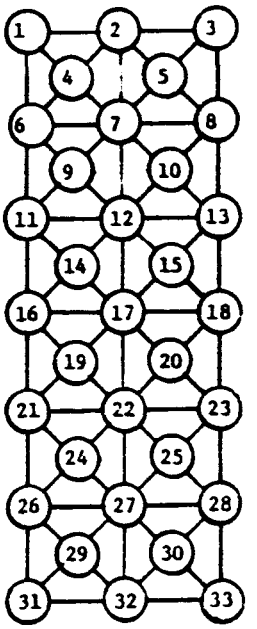


Fig. 2 A 33 node structural problem

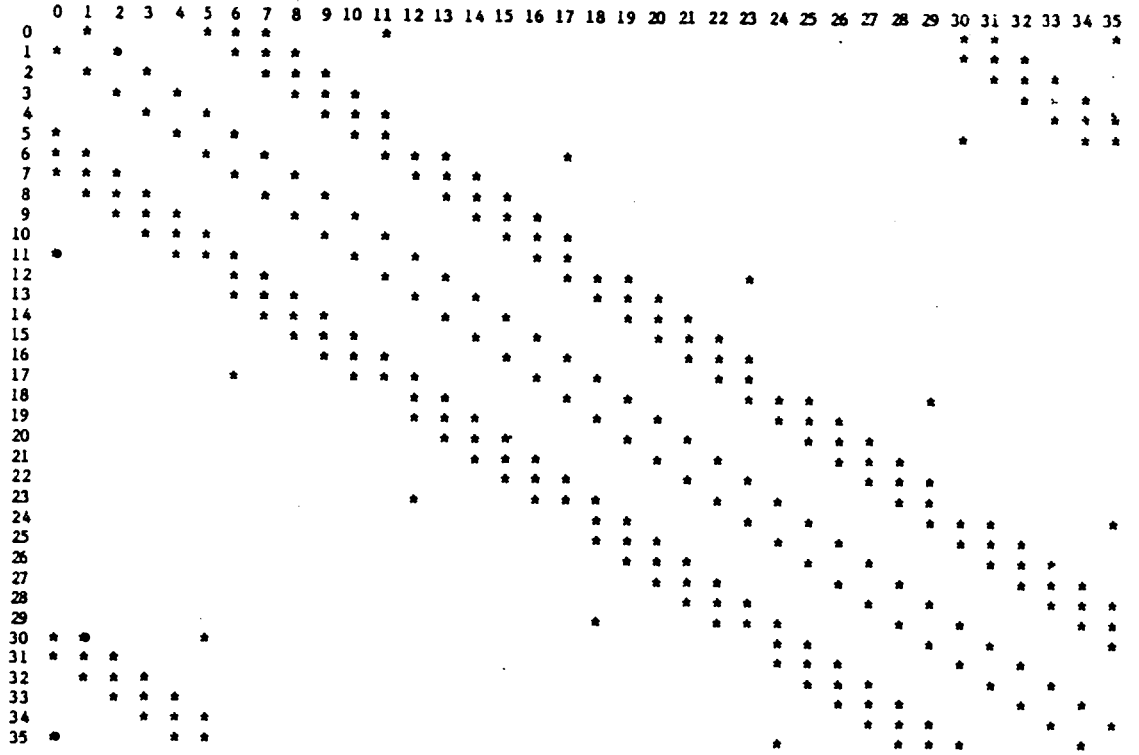


Fig. 3 Adjacency Matrix for the 6 X 6 FEM shown above.

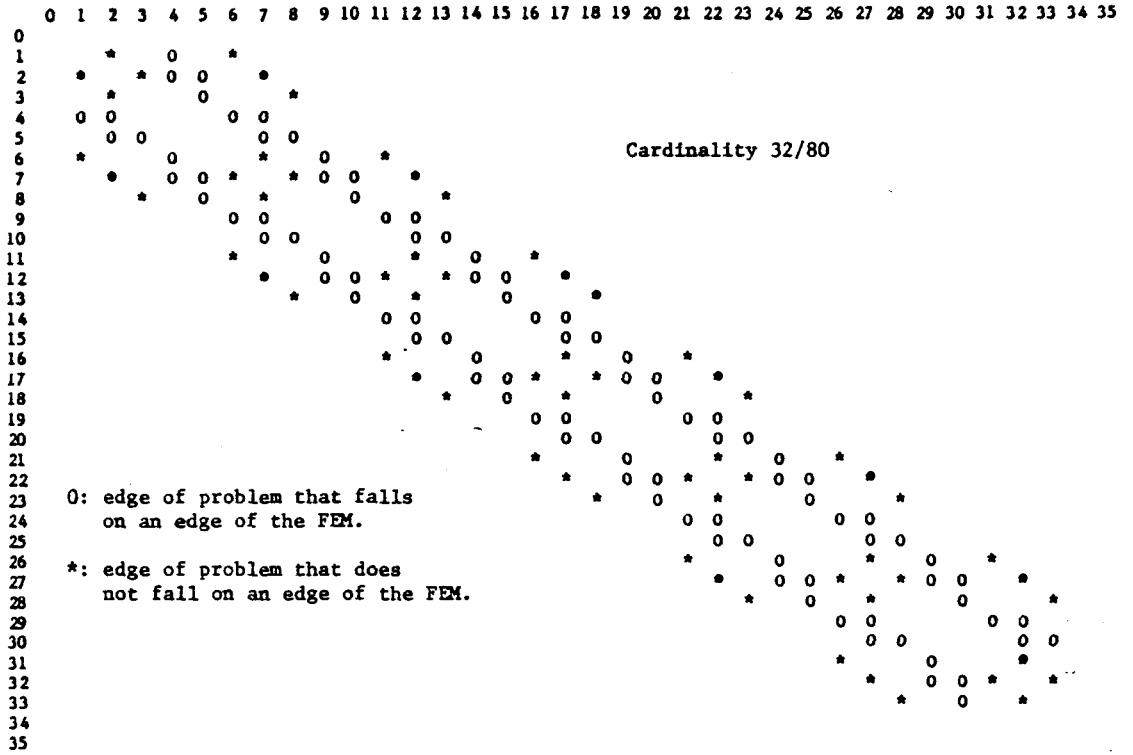


Fig. 4 Initial mapping of the structure shown in Fig. 2 onto the machine of Fig. 1
Above: Initial Mapping. Below: Improved Mapping.

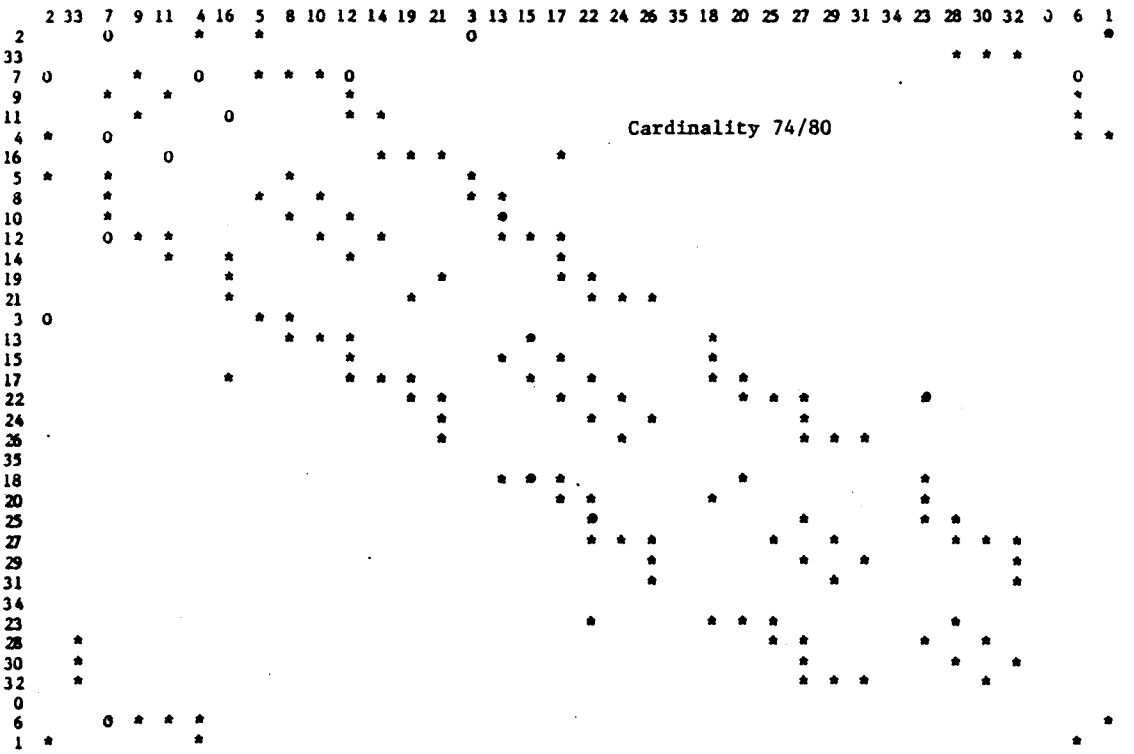


TABLE I

PROBLEM		FEM	TIME	NUMBER OF EXECUTIONS OF		CARDINALITY		EDGES
NAME	NODES	SIZE	MS.	AUGMENT	JUMP	INITIAL	FINAL	
STRUCTURE WITH FREE NODES	33	6X6	11714	14	2	32	74	80
TRUSS	8	4X4	419	5	1	9	15	15
		5X5	1472	5	1	6	15	
		6X6	5083	6	1	6	15	
		7X7	16814	8	1	6	15	
SHIP RADAR TOWER	25	5X5	2912	10	2	44	53	65
SCHWEDLER DOME	6	4X4	321	4	1	6	10	10
+WING BOX	30	6X6	3380	4	1	66	66	78
*NTF-DOWNSTREAM NACELLE	35	6X6	4165	5	1	48	51	58
NTF-NACELLE GUSSET PLATE	39	7X7	29219	14	3	28	49	52
NTF-DOWNSTREAM NACELLE	28	6X6	4224	5	1	37	46	51
NTF-NACELLE BULKHEAD	30	6X6	12650	15	3	27	48	49
NTF-CRADLE	33	6X6	6623	8	1	22	48	52

+The algorithm was unable to improve the given mapping.
*National Transonic Facility.

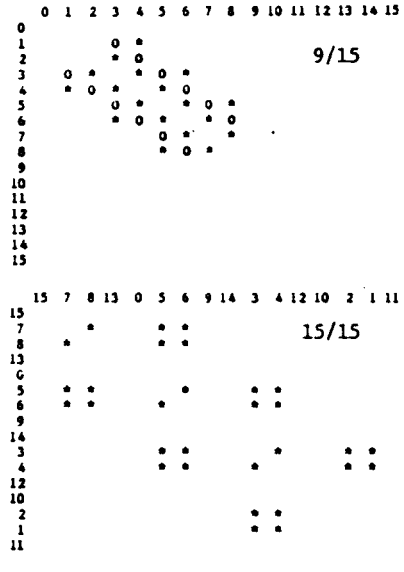


Fig. 5 (Above) Mapping an eight node Truss onto a 4 X 4 FEM

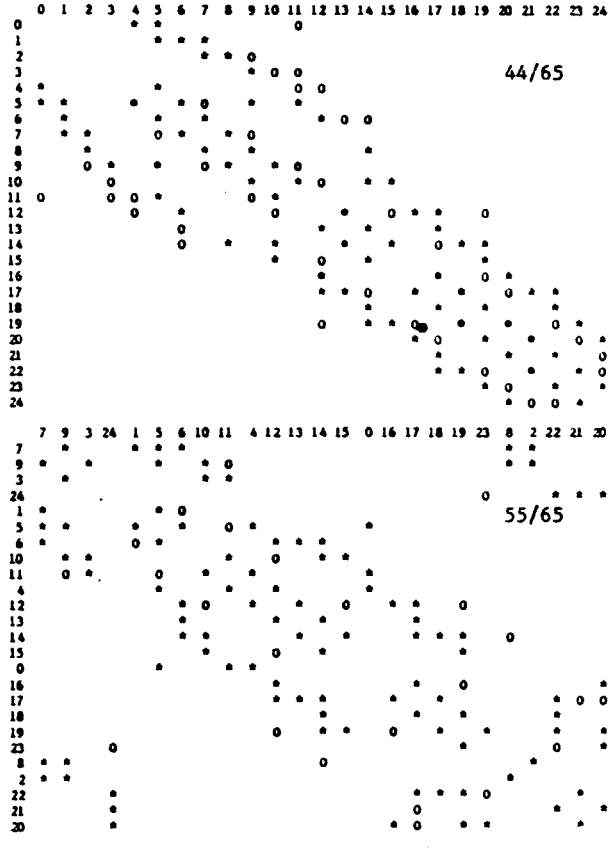


Fig. 6 (Right) 25 node Ship Radar Tower mapped onto 5 X 5 FEM

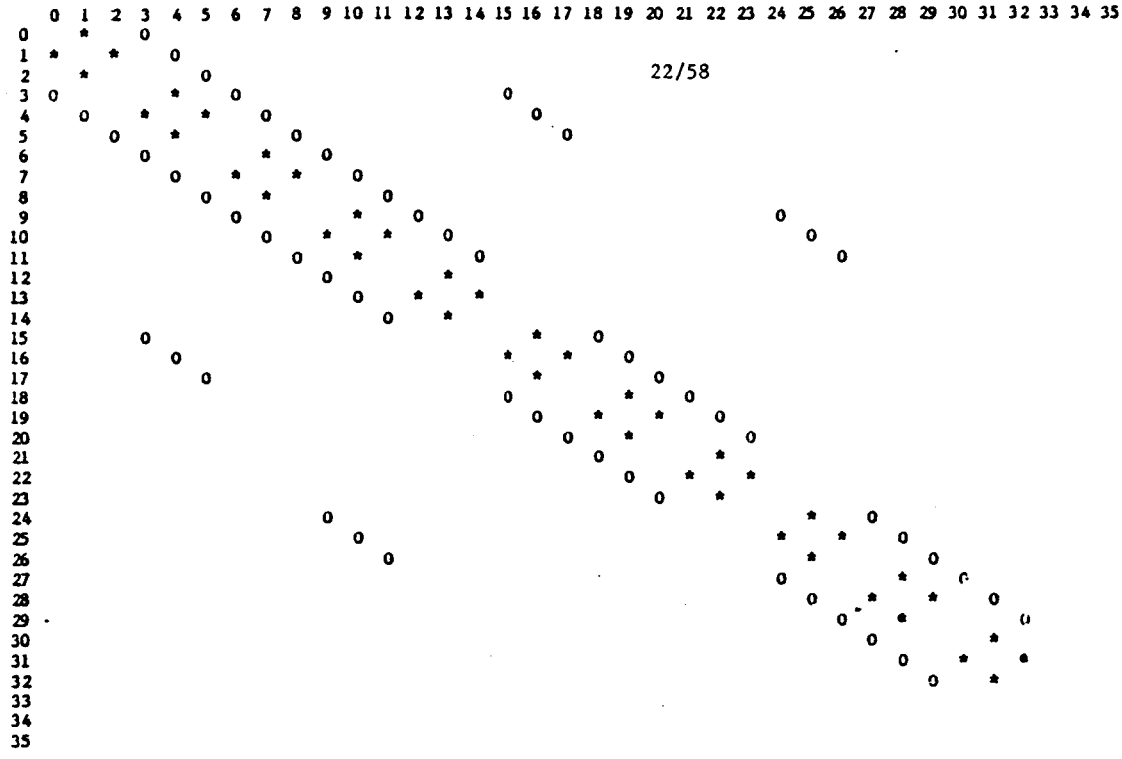
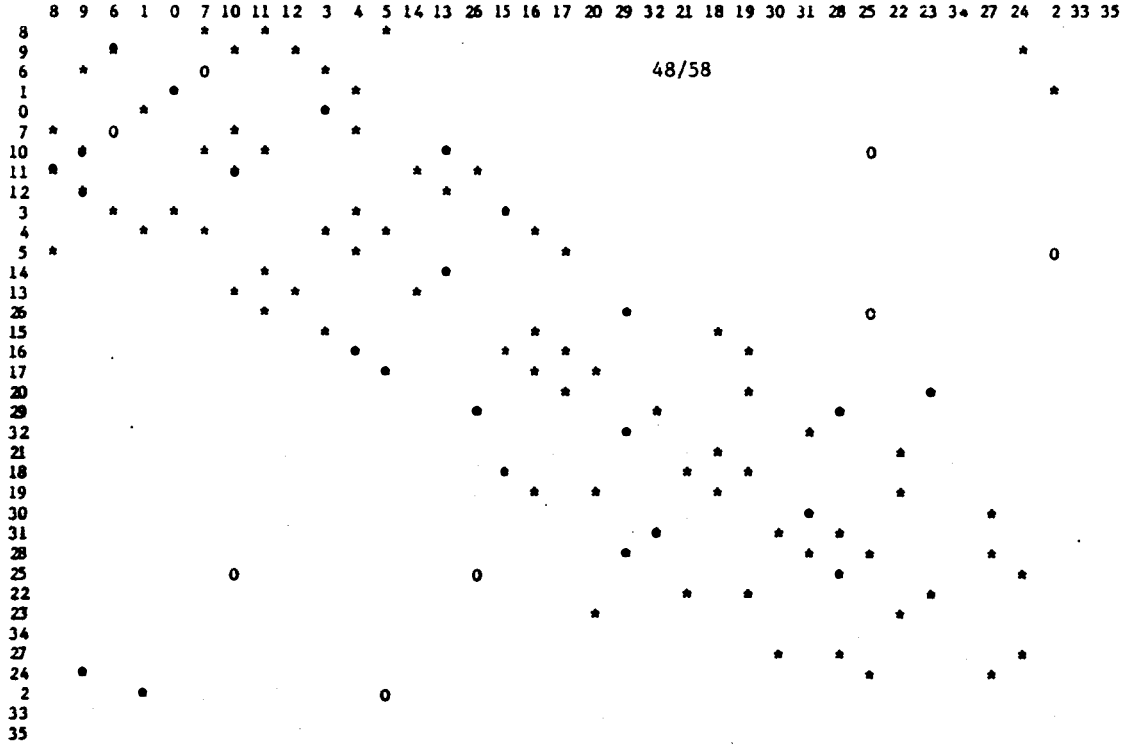


Fig. 7 NTF-Cradle (33 nodes) mapped onto 6 X 6 FEM. Above: initial ; below: final.



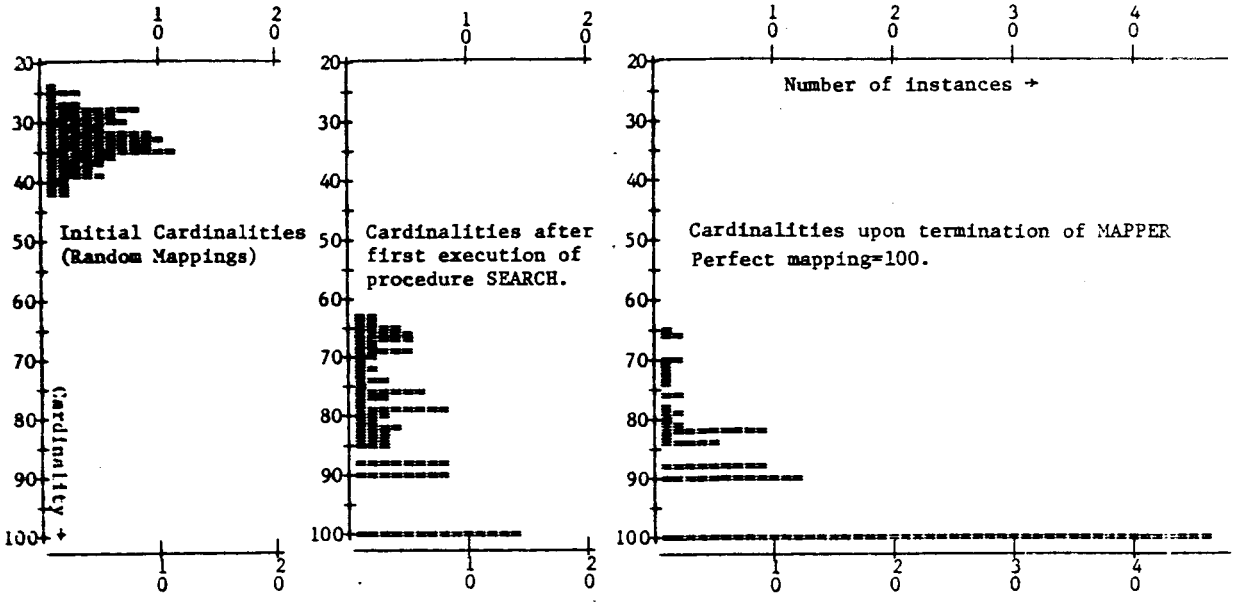


Fig. 8 Histograms showing distributions of cardinalities when 100 randomly permuted FEMs are mapped onto a FEM. Above: 5 X 5 FEMs. Below: 6 X 6 FEMs.